07

# Haga lo que hicimos en aula: otros tópicos

Por último vamos a mostrar 3 recursos que nos permiten organizar aún mejor nuestro código.

> El primero es la anotación `@Embeddable` y `@Embedded` que nos permiten agrupar un conjunto de atributos o propiedades dentro de otra clase que no va a ser indicada como una nueva entidad, sino como una clase que va a ser inyectada dentro de nuestra clase entidad. Con esto podemos agrupar atributos como dirección, datos personales, datos familiares, entre otros, sin tener que colocar millones de atributos en una única clase.

Solo hay que notar que al reemplazar los atributos por una clase embutida tenemos que cuidar que los getter y setter así como, otros métodos continúen funcionando correctamente. De lo contrario podemos delegar la función de retornar esa propiedad para la nueva clase construyendo así un método delegate.

```
@Entity
@Table(name="clientes")
public class Cliente {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    @Embedded
    private DatosPersonales datosPersonales;
```

```
    public Cliente() {}

    public Cliente(String nombre, String dni) {
        this.datosPersonales=new DatosPersonales(nombre,dni);
    }

    public Long getId() {
        return id;
    }
    public String getNombre() {
        return datosPersonales.getNombre();
    }
    public void setNombre(String nombre) {
        this.datosPersonales.setNombre(nombre);
    }
    public String getDni() {
        return datosPersonales.getDni();
    }
    public void setDni(String dni) {
        this.datosPersonales.setDni(dni);
    }
}

@Embeddable
public class DatosPersonales implements Serializable{
    private static final long serialVersionUID = 8063180201812

    private String nombre;
    private String dni;

    public DatosPersonales() {    }

    public DatosPersonales(String nombre, String dni) {
```

```
        this.nombre = nombre;
        this.dni = dni;
    }
//getter and setters
```

COPIA EL CÓDIGO

El segundo recurso es el mapeamiento de herencia, puede que querramos crear nuevas entidades que compartan propiedades o más especificamente que deriven de una clase madre. Para lograr esto no tenemos que colocar todas la propiedades en la nueva entidad, sino que solo debemos marcar la clase madre con la anotación `@Inheritance` y construir nuestras nuevas entidades, normalmente, colocando solo las nuevas propiedades y usando la palabra reservada extends para indicar herencia de clase.

- Clase madre:

```
@Entity
@Table(name="productos")
@NamedQuery(name="Producto.consultarPrecioPorNombre", query="Sl
@Inheritance(strategy=InheritanceType.JOINED)
public class Producto{
…
```

COPIA EL CÓDIGO

- Clase derivada libros:

```
@Entity
public class Libros extends Producto{
```

```
    private String autor;
    private int paginas;
…
```

COPIA EL CÓDIGO

- Clase derivada electrónico:

```
@Entity
@Table(name="electronicos")
public class Electronico extends Producto{
    private String marca;
    private String modelo;
    public Electronico() {    }
    public Electronico(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
    }
…
```

COPIA EL CÓDIGO

Y ya por último, existe un recurso que nos permite identificar elementos en nuestras entidades utilizando más de un único parámetro, generalmente utilizamos un parámetro que puede ser numérico o alfanumérico para identificar las filas. A esto se llama llave primaria única, pero podemos utilizar llaves primarias con parámetros compuestos, para eso JPA nos provee como recurso la anotación @EmbeddedId que nos permite indicarle a nuestra entidad que está siendo inyectada una clase que no es una entidad, sino una clase que compone el ID de esa entidad formado por multiples parametros.

```
@Entity
@Table(name="categorias")
public class Categoria {

    @EmbeddedId
    private CategoriaId categoriaId;

    public Categoria() {}
```

COPIA EL CÓDIGO

- En la clase que va a componer la llave primaria la anotamos con @Embeddable como lo habíamos hecho anteriormente, pero la anotación `@EmbeddedId` va a indicarle a la entidad que debe tomar esa clase como ID, adicional todas las clases embutidas sean llaves primarias o no deben implantar la interfaz Serializable ya que sirve para indicarle a la API que van a haber datos transitando dentro de ella.

```
@Embeddable
public class CategoriaId implements Serializable{

    private static final long serialVersionUID = 4198020985304

    private String nombre;
    private String password;
    public CategoriaId() {
    }
    public CategoriaId(String nombre, String password) {
        this.nombre = nombre;
        this.password = password;
    }

//getters setters
```

COPIA EL CÓDIGO